Performance Monitoring and Analysis Components in Adaptive PDE-Based Simulations [1]

B. Norris

Mathematics and Computer Sciences Division,

Argonne National Laboratory, 9700 South Cass Ave.,

Argonne, IL 60439-4844

I. Veljkovic

Department of Computer Science and Engineering, The Pennsylvania State University, 220 Pond Lab, University Park, PA 16802-6106

Abstract

As scientists incorporate more sophisticated models into their simulations, software complexity, as well as the underlying computational cost of these models, is growing rapidly. Performance evaluation and tuning of applications that are large-scale both in terms of source code and runtime requirements can be challenging and time-consuming for scientists. We have developed a software infrastructure for performance monitoring, performance data management, and adaptive algorithm development for parallel component PDE-based simulations. We have instrumented

Newton-Krylov nonlinear and linear solver components for performance monitoring using the TAU performance tools. To reduce the performance monitoring and component adaptation overhead, we employ two databases. The first is created and destroyed during runtime and stores performance data for code segments of interest, as well as various application-specific performance events in the currently running application instance. The second database is persistent and contains performance data from various applications and different instances of the same application. It can also contain performance information derived through offline analysis of raw data. We describe a prototype implementation of this infrastructure and show how adaptive linear solver algorithms are employed in a driven cavity flow simulation code.

Key words: performance analysis, components, adaptive methods, simulation

1 Introduction

Our performance infrastructure is motivated by the needs of parallel simulations based on the solution of partial differential equations (PDEs), such as computational fluid dynamics, fusion, accelerator design, climate modeling, and combustion. PDE-based scientific software complexity has been increasing

Email addresses: norris@mcs.anl.gov (B. Norris), veljkovi@cse.psu.edu (I. Veljkovic).

[1] This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

as new models and solution methods emerge. Often a single simulation code involves multiphysics, multimodel, multidisciplinary, and multi-institutional software development. Typical numerical methods for such problems incorporate the solution of large sparse linear systems of equations. Solution of such equations typically takes 70%–85% of the total simulation time and is therefore a logical candidate for extensive performance analysis and optimization. New, more robust or efficient algorithms are the traditional approach to achieving better performance. Another approach is to develop *multimethod* linear solvers [1–5] that involve the application of several algorithms in the course of solving the same problem. Several different types of multimethod approaches exist; in this paper, we focus on two of them: composite and adaptive methods. A *composite* linear solver comprises several underlying (base) methods, which are applied in sequence to the *same* linear system until convergence is achieved. Such solvers are normally constructed statically, by using past base method performance data and a simple performance model combining robustness and execution time. An *adaptive* linear solver applies a sequence of base solution methods to *different* linear systems arising at each nonlinear iteration of the PDE solution. Adaptive heuristics usually select one among several candidate base methods at runtime in order to optimize some performance attribute of an application, for example, an approximation of the nonlinear convergence rate or total simulation time. Detailed descriptions of these multimethod approaches can be found in [1–3].

While performance analysis-based algorithm selection and adaptation have produced some promising results in terms of application performance, the development of multimethod strategies is complicated and time-consuming because of the lack of a software infrastructure supporting such nontraditional methods. This has motivated the design and implementation of the component performance monitoring and analysis interfaces and components described in this paper. Our design was aided and influenced by our ongoing research in *computational quality of service* (CQoS) [6, 7], which involves the automatic selection and configuration of components to suit a particular computational purpose.

We now introduce some of the terminology used in this article. We collectively refer to performance-relevant attributes of a unit of computation, such as a component, as performance *metadata*, or just metadata. These attributes include algorithm or application parameters, such as problem size and physical constants, compiler optimization options, and execution information, such as hardware and operating system information. Performance metrics, also referred to as CQoS metrics, are also part of the metadata, for example, execution time and convergence history of iterative methods. Ideally, for each application execution, the metadata should provide enough information to be able to repeat the run; we collectively refer to these metadata as an application instance, or *experiment*. The remainder of this section briefly introduces the technologies that provide the specifications and tools that have enabled

the creation of our performance monitoring and analysis infrastructure.

1.1 Component technology for scientific computing

The Common Component Architecture Forum [8–10] was launched in 1998 as a grassroots initiative to bring the benefits of component-based software engineering to high-performance scientific computing. In 2001, the U.S. Department of Energy (DOE) established the Center for Component Technology for Terascale Simulation Software (CCTTSS) [11], which supports component research and software infrastructure development. As in other component-based software engineering approaches, such as CCM [12], EJB [13], and COMM [14], the goals are to help manage software complexity and to enable reuse of components in multiple applications. Unlike these commodity component models, the CCA specifically targets high-performance applications, particularly their need for low-overhead component interactions, parallelism, and support for language interoperability between components implemented in languages typically used in scientific software development, such as Fortran, C, and C++. This approach ensures minimal overhead in most cases (approximately the cost of a virtual function call) for component interactions and possible data type conversions handled by the Babel language interoperability layer [15, 16] used in CCA [17].

Briefly, CCA components are units of encapsulation that can be composed to

form applications; *ports* are the entry points to a component and represent public interfaces through which components interact; *provides* ports are interfaces that a component implements, and *uses* ports are interfaces that a component uses. A runtime framework provides some standard services to all CCA components, including instantiation of components, and *uses* and *provides* port connections. Components can be instantiated/destroyed and port connections made/broken at runtime, thereby allowing dynamic adaptivity of CCA component applications and enabling the implementation of the adaptive linear solver methods introduced above.

1.2 Performance tools

Tuning and Analysis Utilities (TAU) [18, 19]. TAU is a portable profiling and tracing toolkit for performance analysis of parallel programs. In addition to providing a portable instrumentation interface, TAU can be used in conjunction with the Program Database Toolkit [20] to instrument code automatically at the function level. A TAU-based abstract interface for scientific components was also recently introduced [21] and used in constructing performance models in component applications [7, 22], as well as in our implementation as described in Section 3.

Performance Data Management Framework (PerfDMF) [23]. PerfDMF is a new, general-purpose environment for performance data management, in-

cluding importing and exporting of data from parallel profiling tools, portable large-scale profile data management, and abstract interfaces for database access. In our work we mainly use the profile database component, which supports a number of database engines, including PostgreSQL, MySql, Oracle, and DB2.

The rest of this paper is organized as follows. Section 2 describes our approach to designing a component infrastructure that supports performance monitoring, analysis, and adaptation. Section 3 presents implementation details of this infrastructure. Section 4 illustrates the use of our infrastructure in a driven cavity flow simulation, and Section 5 contains our conclusions, as well as brief discussion of ongoing and future work.

2 Performance monitoring and analysis infrastructure

In high-performance scientific computing, execution time is one of the key parameters when considering performance, but it is not the only one. Other application attributes can affect the *quality* of a scientific simulation, such as convergence rate, stability, parallel scalability, and accuracy of the final result. These are not independent of each other, and optimizing all simultaneously is difficult or impossible. For example, maximizing accuracy will almost certainly result in longer execution time. We consider these and other application-specific CQoS parameters when discussing the *performance* of component

applications in general. Our goal is to enable the implementation of CQoS-enhanced applications by providing the necessary performance data gathering and manipulation infrastructure, which can then be used by algorithms and heuristics to automate component application assembly and support dynamic adaptation based on CQoS metadata from analytical performance models or synthesized from past performance history of the application.

For a particular application, such as the driven cavity flow example introduced in Section 4, several parameters (e.g., grid size, lid velocity, Grashof number, or initial CFL number) can be varied to create problem instances of varying difficulty. Every combination of parameters represents a certain experiment, or instance of a problem. As Figure 1 illustrates, we can loosely divide these parameters into three categories: *model parameters*, which influence the definition of the problem, such as lid velocity in the driven cavity problem; *algorithmic parameters*, which characterize the implementation approach, such as error tolerance for a nonlinear solver; and *architecture-specific compiler and hardware parameters*, which can influence the execution time of an application. Our initial focus is on model and algorithmic parameters, with plans to incorporate architecture-specific parameters later. One of the main goals of our framework is to enable and support the development of performance-improving component assembly and adaptation strategies based on performance analysis results annotated with CQoS parameters.

Finding a set of algorithms that are best for solving a problem or a part

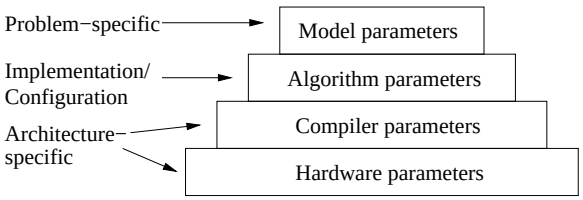

Fig. 1. Hierarchy of parameters for a given application. Some (e.g., problem size) are fixed, while others (e.g., algorithmic parameters or compiler options) can be fine-tuned for better performance.

of a problem is a difficult task. For example, there are numerous choices for iterative linear solvers for large sparse systems; since properties of the linear systems change, it would be desirable to use a solver that achieves the best performance on the current system. Most iterative algorithms, however, do not come with neat performance models that provide this information a priori.

An alternative approach is to analyze past performance information and try to identify those CQoS parameters that have the greatest impact on performance. One of our ultimate objectives is to discover whether there are common performance characteristics for given CQoS parameters across various experiments. If such characteristics exist, we would like to represent them in the performance database in the form of metadata. One can then develop adaptive strategies that take advantage of the derived metadata in addition to current execution data to choose a best set of parameters and the best algorithm for a given experiment.

The design of our infrastructure was guided by the following goals: (1) low overhead during the application's execution; since all the time spent in performance monitoring and analysis/adaptation is overhead, the impact on overall

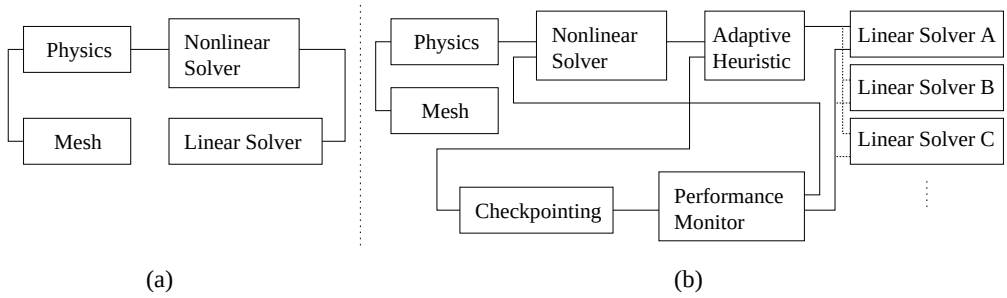

Fig. 2. Some of the components and port connections in a typical PDE application: (a) in a traditional nonadaptive setting, and (b) augmented with performance monitoring and adaptive linear solver components.

performance must be minimized; (2) minimal code changes to existing application components in order to encourage use of this performance infrastructure by as many CCA component developers as possible; and (3) ease of implementation of performance analysis algorithms and new adaptive strategies, which would enable and encourage the development and testing of new heuristics or algorithms for multimethod components.

Figure 2 (a) shows a typical set of components involved in nonlinear PDE applications; no explicit performance monitoring or adaptive method support is available. Figure 2 (b) shows the same application with the new performance infrastructure components.

The adaptive heuristics component implements a simple *AdaptiveAlgorithm* interface, whose single method, `adapt`, takes an argument containing application-specific metadata needed for implementing a particular adaptive heuristic and storing the results. Specific implementations of the *AdaptiveContext* interface contain performance metadata used by adaptive heuristics, as well as references to the objects that provide the performance metadata contained in the

context.

Within our framework we have to differentiate between tasks that have to be completed at runtime and tasks that are performed when the experiment is finished. Consequently, we have two databases that serve significantly different purposes. The first one is created and destroyed during runtime and stores performance data for code segments of interest and application-specific performance events for the running experiment. The second database is persistent and contains data about various applications and experiments within one application. The second database also contains metadata derived by performance analysis of raw performance results. At the conclusion of an experiment, the persistent database is updated with the information from the runtime database.

2.1 Runtime support

Figure 3 illustrates the components involved in the dynamic performance monitoring and analysis. The "Numerical Component" represents any of the components involved in the application for which monitoring and optionally adaptation can be done. Our initial focus is on adaptive linear solvers in the context of nonlinear PDE solution; therefore, the "Numerical Component" represents the nonlinear and linear solver components. The following components provide performance monitoring and data management support at runtime.

- **TAU Measurement Component**. This component collects runtime data from hardware counters, timing, and user-defined application-specific events. This component was provided by the developers of TAU, and complete implementation details can be found in [22].
- **Checkpoint Component**. This component checkpoints and stores the collected data into a runtime database that can be queried efficiently during the execution for the purpose of runtime performance monitoring and adaptation. The TAU profiling API can only give either callpath-based or cumulative performance information about an instrumented object (from the time execution started). Hence, we have introduced the Checkpoint component to enable us to store and retrieve data for the instrumented object during the application's execution (for example, number of cache misses for every three calls of a particular function). The period for checkpointing can be variable; the component can also be used by any other component in the application to collect and query context-dependent and high-level performance information. For example, a linear solver component can query the checkpointing component for performance metadata of the nonlinear solver (the linear solver itself has no direct access to the nonlinear solver that invoked it). We can therefore always get the latest performance data for the given instrumented object from the database constructed during runtime.
- **Metadata Extractor**. This component retrieves metadata from the database at runtime. After running several experiments, analyzing the performance data, and finding a common performance behavior with some pa-

rameter values, we store data summarizing this behavior in the database. An example of derived metadata is the rate of convergence of a nonlinear or a linear solver. During runtime, these data are used in adapting our parameter and algorithm selection, and the Metadata Extractor component can retrieve compact metadata from the database efficiently.

- **Monitor Component**. This component monitors the application and the algorithm and parameter selection based on runtime performance data and stored metadata.

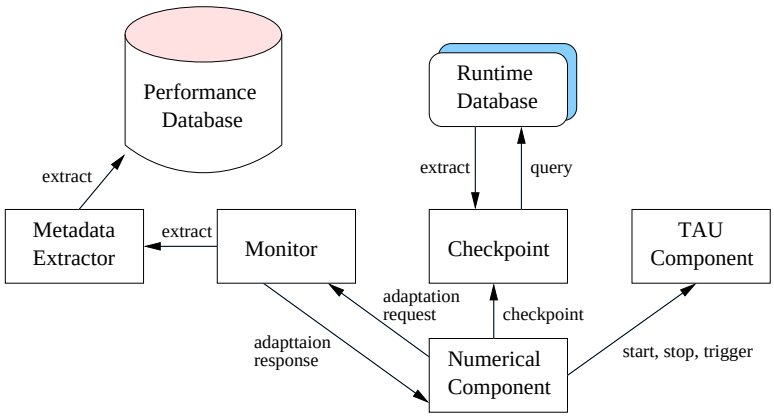

Fig. 3. Dynamic performance components.

2.2 Offline analysis support

The portions of the infrastructure that are not used at runtime are illustrated in Figure 4. They include a performance data extractor for retrieving data from the performance database, which is used by the offline analysis algorithm components. At present, the extractor also produces output in Matlab-like format, which is convenient for plotting some performance results; this output can be enhanced to interface with tools that provide more advanced visualization ca-

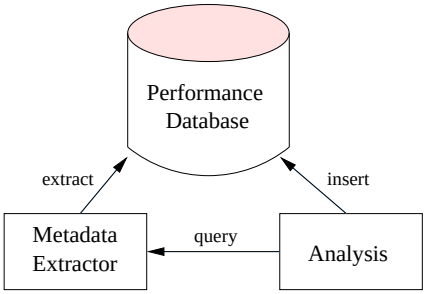

Fig. 4. Offline analysis components.

pabilities, such as an extension of ParaProf (part of the TAU suite of tools). Many analyses can be applied offline to extract performance characteristics from the raw execution data or the results of previous analyses—in fact, facilitating the development of such analyses was one of our main motivations for developing this performance infrastructure. Initially we are focusing on simple analyses that allow us to replicate results in constructing composite linear solvers from performance statistics of base linear solver experiments [1]. For the longer term, we plan to use this infrastructure for rapid development of new performance analyses and adaptive heuristics.

3 Implementation

Our initial implementation consisted only of C++ components and C libraries. Since many scientific applications are written in Fortran, we next developed language-independent interfaces and corresponding initial implementations using the Scientific Interface Definition Language (SIDL) and Babel [15]. Our current implementation is still in C++ but is accessible from other languages through the Babel-generated stubs code, which automates the interoperability

of object-oriented codes written in C, C++, Fortran, Python, and Java.

Our persistent database is based on the PerfDMF specification, with small extensions. Since we are collecting performance data for iterative algorithms, where different iterations do not normally take the same amount of time, we need to express and store performance information for single iterations or ranges of iterations. While TAU allows callpath profiling of an application, this approach is too general for our purposes and imposes too great an overhead. Our performance metadata table has the following fields:

- Iteration range for which the metadata result applies (this is a two-element, one-dimensional array)
- Average slope of change of metadata in this range
- Input parameter range for which the metadata result applies (this is a two-dimensional array).

This information enables the user or analysis algorithm to investigate certain behavior for a particular subset of experiments. The component that performs the data analysis must ensure that, within this range of iterations and parameters, the minimum and maximum slope do not vary much from the average slope. Currently the runtime interface supports two types of queries of the performance database:

- *Extracting metadata for a certain performance parameter.* Metadata are related to a common behavior of performance values recorded with TAU

over a wide range of experiments performed on an application. For example, we may want to find out for which nonlinear iterations the slope of CFL number is bigger than some predefined number. We provide a C++ interface and corresponding library for easy access to this kind of information. This library is a thin wrapper over the SQL API and therefore can be used for arbitrary SQL queries and is not limited to metadata extraction.

- *Finding "optimal" algorithm parameters.* In some cases we wish to find a value for a certain algorithmic parameter that yields the best performance over the experiments that have already been profiled. Algorithmic parameters can be fine-tuned in order to achieve better performance. For example, we may want to find out what fill level for incomplete factorization preconditioning would yield the fastest overall convergence of a nonlinear solver. The user is not limited to specifying only one performance metric: if more than one is given, then the parameter value optimizes the weighted sum of the performance metric values computed with user-specified weights. For each of the options, the user can search for parameter values over all experiments for a given application or only for experiments that have certain model parameters in a certain range (for example, in the driven cavity application example in Section 4, one can specify that grid size should be larger than 100 or lid velocity should be 130). The output is a matrix in which the first column is the parameter value and the rest of the columns contain the values of the performance metrics specified. Figure 5 illustrates an example use of this query interface.

```
//NOTE: Name of a particular function is recorded in the database
//as name+type.
string function_name("main() int (int, char **)");
string metric_name("PAPI_FP_INS");
string param_name(" initial_cfl");
bool inclusive = true;

ierr = det.DetermineParameterMetric(param_name,
                                    metric_name, function_name,
                                    inclusive, &result_size, &result,
                                    minimum, &optimal_index,
                                    dstart, dend, lstart, lend);
```

Fig. 5. Code segment for extracting optimal values for algorithm parameters from the database.

4 Application example

We illustrate the use of our performance infrastructure in a computational fluid dynamics application that simulates flow in a driven cavity, which combines lid-driven flow and buoyancy-driven flow in a two-dimensional rectangular cavity. We use a velocity-vorticity formulation of the Navier-Stokes and energy equations, which we discretize using a standard finite-difference scheme with a five-point stencil for each component on a uniform Cartesian mesh; see [24] for a detailed problem description. Commonly used pseudo-transient continuation methods introduce a false time-stepping term into the model and necessitate solving a nonlinear system of equations at each time step using Newton's method. The transition of the time step from small to large controls the conditioning of the linearized Newton systems; thus, the resulting linear systems are initially well-conditioned and easy to solve, while later in the simulation they become progressively harder to solve.

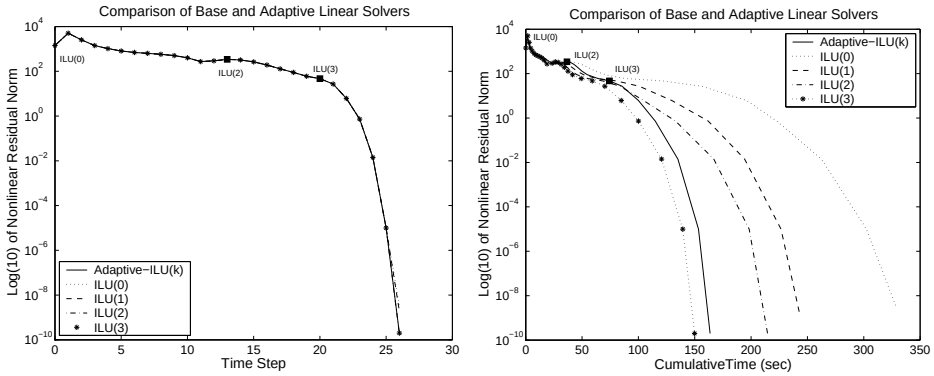

Fig. 6. Comparison of single-method linear solvers and an adaptive scheme. We plot the nonlinear convergence rate (in terms of residual norm) versus both time step (left-hand graph) and time (right-hand graph).

In this parallel application, metadata describing the performance of the nonlinear solution, as well as each linear solution method, can be used to determine when to change or reconfigure linear solvers. More details about the adaptive methodology can be found in [2, 3]. Figure 6 shows some performance results comparing the use of an adaptive heuristic with the traditional single solution method approach. The automated adaptive strategy performs better than most base methods, and almost as well as the best base method (whose performance, of course, is not known a priori).

Another use of our performance infrastructure is selection of application parameters based on performance information with the goal of maximizing performance. For example, in the application considered here, the initial CFL value is essential for determining the false time step for the pseudo-transient Newton solver, which in turn affects the overall rate of convergence of the problem. Using the query interface described in Section 3, we can query the database to determine the best initial CFL value from the experimental data

available. We hope that this CFL value will perform better than a random CFL value or at least as well as the CFL value adopted by other researchers when performing the experiments; we plan to evaluate this in further experiments.

5 Conclusions and future work

We have designed a framework for performance monitoring, evaluation, and adaptation of parallel scientific applications and implemented a prototype using CCA components, TAU, and the PerfDMF database format. This infrastructure enables the performance characterization of scientific component applications and the rapid development of performance analyses and adaptive algorithms. We illustrated our initial implementation with an application that represents the problem domain we are targeting.

Our current and future work includes expanding our current implementation with more components for offline analyses of performance information, as well as developing new adaptive heuristics for dynamic method selection. In addition to runtime adaptation, our performance infrastructure can support initial application assembly and can potentially be integrated with existing CCA component infrastructure that uses component performance models for automated application assembly [7, 22].

We have performed some initial tests with the driven cavity application and will next consider other PDE-based simulations with similar structure, such as

compressible Euler flow. A large number of experiments will be performed to populate the database with enough performance data for analyses algorithms. One of the short-term goals is to use the new performance infrastructure in the automatic definition of composite linear solvers based on single method performance data (currently composite algorithms are assembled manually). We will also extend our infrastructure with new types of performance meta-data as we develop new analyses. One longer-term objective is to investigate whether accurate performance models can be synthesized or refined through statistical analysis of the parameterized performance information of applications, potentially with the help of models derived through source code analysis of the numerical component implementations.

References

[1] S. Bhomick, P. Raghavan, L. McInnes, , B. Norris, Faster PDE-based simulations using robust composite linear solvers, Future Generation Computer SystemsAccepted for publication. Also published as a technical report, Argonne National Laboratory, ANL/MCS-P993-0902.

[2] S. Bhowmick, L. C. McInnes, B. Norris, P. Raghavan, The role of multi-method linear solvers in PDE-based simulations, Lecture Notes in Computer Science, Computational Science and Its Applications-ICCSA 2003 2667 (2003) 828–839.

[3] L. McInnes, B. Norris, S. Bhowmick, P. Raghavan, Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms, Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics.

[4] R. Barrett, M. Berry, J. Dongarra, V. Eijkhout, C. Romine, Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach, J. Comp. Appl. Math. 74 (1996) 91–109.

[5] J. R. Rice, On the construction of poly-algorithms for automatic numerical analysis, in: M. Klerer, J. Reinfelds (Eds.), Interactive Systems for Experimental Applied Mathematics, Academic Press, 1968, pp. 301–313.

[6] P. Hovland, K. Keahey, L. C. McInnes, B. Norris, L. F. Diachin, P. Raghavan, A quality of service approach for high-performance numerical components, in: Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference, Toulouse, France, 2003.

[7] B. Norris, J. Ray, R. Armstrong, L. C. McInnes, D. E. Bernholdt, W. R. Elwasif, A. D. Malony, S. Shende, Computational quality of service for scientific components, in: Proceedings of the International Symposium on Component-Based Software Engineering, (CBSE7), Edinburgh, Scotland, 2004.

[8] Common Component Architecture Forum, Common Component Architecture (CCA), `http://www.cca-forum.org` (2004).

[9] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, S. Zhou, A component architecture for high-performance scientific computing, submitted to *Intl. J. High-Perf. Computing Appl.* (2004).

[10] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, B. Smolinski, Toward a Common Component Architecture for high-performance scientific computing, in: Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, 1999.

[11] CCTTSS, DOE SciDAC Center for Component Technology for Terascale Simulation Software, `http://www.cca-forum.org/ccttss` (2005).

[12] Object Management Group, CORBA component model, `http://www.omg.org/technology/documents/formal/components.htm` (2002).

[13] Sun Microsystems, Enterprise JavaBeans downloads and specifications, `http://java.sun.com/products/ejb/docs.html` (2004).

[14] Microsoft Corporation, Component Object Model specification, `http://www.microsoft.com/com/resources/comdocs.asp` (1999).

[15] T. Dahlgren, T. Epperly, L. L. N. L. Gary Kumferti, Babel Users's Guide (Version 0.8.8) (2003).

[16] S. Kohn, G. Kumfert, J. Painter, C. Ribbens, Divorcing language dependencies from a scientific software library, in: Proceedings of the Workshop on Performance Optimization via High-Level Languages (POHLL-02), 2002.

[17] D. Bernholdt, W. Elwasif, J. Kohl, T. Epperly, A component architecture for high-performance computing, in: Proceedings of the Workshop on Performance Optimization via High-Level Languages (POHLL-02), 2002.

[18] Anonymous, TAU User's Guide (Version 2.13) (2004).

[19] J. Dongarra, A. D. Malony, S. Moore, P. Mucci, S. Shende, Performance instrumentation and measurement for terascale systems, in: Proc. of the International Conference on Computational Science (ICCS), Vol. 2660 of Lecture Notes in Computer Science, Springer, Berlin, 2003, pp. 53–62.

[20] K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, C. Rasmussen, A tool framework for static and dynamic analysis of object-oriented software with templates, in: Proceedings of SC2000: High Performance Networking and Computing Conference, 2000.

[21] S. Shende, A. D. Malony, C. Rasmussen, M. Sottile, A performance interface for component-based applications, in: Proc. of the International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems, IPDPS'03, IEEE Computer Society, 2003.

[22] A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, M. Sottile, Performance technology for parallel and distributed component software, Concurrency and Computation: Practice and Experience 17 (2005) 117–141.

[23] K. Huck, A. Malony, R. Bell, L. Li, A. Morris, PerfDMF: Design and implementation of a parallel performance data management framework, 2005, submitted.

[24] T. Coffey, C. Kelley, D. Keyes, Pseudo-transient continuation and differential algebraic equations, SIAM J. Sci. Comp 25 (2003) 553–569.